Laboratorio di Analisi Numerica
Soluzioni degli esercizi

Federico Poloni <f.poloni@sns.it>

10 gennaio 2011

Lezione 1

```
function p=pow(x,n)
%calcola x^n, per n intero nonnegativo
p=1;
for k=1:n
 p=p*x;
endfor
endfunction
```

```
function s=myexp3(x,n)
s=1;
t=1; %accumulatore per il k-esimo termine della serie di Taylor
for k=1:n
 t=t*abs(x)/k;
 s=s+t;
endfor
if(x<0)
 s=1/s;
endif
endfunction
```

```
function [x1 x2]=solve2(a,b,c)
d=sqrt(b^2-4*a*c);
if(d*b>0)
  x1=(-b-d)/(2*a);
else
  x1=(-b+d)/(2*a);
endif
x2=c/(a*x1);
```

```
endfunction
```

```
function y=coslimit(x)
%calcola (1-cos(x))/x^2 in modo accurato anche per x piccoli
 y=2*sin(x/2)^2/x^2; %serve un po' di trigonometria per dimostrarlo...
endfunction
```

```
function sum=kahansum(a,b,c)
%calcola a+b+c con l'algoritmo di sommazione di Kahan
sum=a;
delta=0;

%non abbiamo ancora visto i vettori, quindi facciamo
%la somma dei due valori a mano...
y=b-delta;
t=sum+y;
d=(t-sum)-y;
sum=t;

y=c-delta;
t=sum+y;
d=(t-sum)-y;
sum=t;
endfunction
```

Lezione 2

```
function circle(z,r)
x=real(z);
y=imag(z);
t=0:0.01:2*pi;
plot(r*cos(t)+x,r*sin(t)+y);
endfunction
```

```
function ggsecond(A)
  n=size(A)(1);
  clearplot;
  hold on;
  axis('equal');
  for t=0:.01:1
    autoval=eig((1-t)*A+t*diag(diag(A))); %"help diag" per saperne di piu'
    plot(real(autoval),imag(autoval),'1.');
  endfor
  for k=1:n
    center=A(k,k);
```

```
    radius=0;
    for j=1:n
      if(j~=k)
        radius=radius+abs(A(k,j));
      endif
    endfor
    circle(center,radius);
  endfor
endfunction
```

```
function f=fattoriale_fast(n)
f=prod(1:n); %prodotto degli elementi del vettore [1 2 ... n]
endfunction
```

```
function p=pow_fast(x,n)
p=prod(x*ones(n,1)); %prodotto degli elementi del vettore [x x ... x]
endfunction
```

```
function f=myexp_fast(x,n)
v=x*ones(1,n)./(1:n);
f=1+sum(cumprod(v));
endfunction
```

Lezione 3

```
function d=mydet(A)
 n=size(A)(1);
 if(n==1)
  d=A(1,1);
 else
  d=0;
  for j=1:n
   d=d+(-1)^(1+j)*mydet(minor(A,1,j))*A(1,j);
  endfor
 endif
endfunction
```

```
function d=mydet2(A)
 n=size(A)(1);
 for k=1:n
  if(A(k,k)==0)
   error('pivot nullo');
  endif
  for i=k+1:n
   A(i,:)=A(i,:)-A(i,k)/A(k,k)*A(k,:);
```

```
  endfor
 endfor
 d=prod(diag(A));
endfunction
```

```
function d=mydet3(A)
 n=size(A)(1);
 for k=1:n
  if(A(k,k)==0)
   error('pivot nullo');
  endif
  [useless p]=max(abs(A(k,k:n)));
  p=p+k-1;
  temp=A(p,:); %questa istruzione e le due seguenti scambiano le righe A(p,:) e A(k,:)
  A(p,:)=A(k,:);
  A(k,:)=temp;
  if(p~=k) %solo se c'e' stato un vero scambio di righe...
    A(k,:)=-A(k,:); %compensa il cambio di segno introdotto nel determinante
  endif
  for i=k+1:n
   A(i,:)=A(i,:)-A(i,k)/A(k,k)*A(k,:);
  endfor
 endfor
 diag(A)
 d=prod(diag(A));
endfunction
```

Lezione 4

```
function A=laplacian(n)
for k=1:n
 A(k,k)=2;
endfor

for k=1:n-1
 A(k,k+1)=-1;
 A(k+1,k)=-1;
endfor
endfunction
```

```
function x=sup_solve(U,b)
n=size(U)(1);
x=zeros(n,1);
for i=n:-1:1
 y=b(i);
```

```
 for j=i+1:n
  y=y-U(i,j)*x(j);
 endfor
 %y vale il numeratore
 x(i)=y/U(i,i);
endfor
endfunction
```

```
function x=inf_solve2(L,b)
%risolve sistemi con matrice bidiagonale inferiore
n=size(L)(1);
x=zeros(n,1);
x(1)=b(1)/L(1,1);
for i=2:n
 x(i)=(b(i)-L(i,i-1)*x(i-1))/L(i,i);
endfor
endfunction
```

```
function x=sup_solve2(U,b)
%risolve sistemi con matrice bidiagonale superiore
n=size(U)(1);
x=zeros(n,1);
x(n)=b(n)/U(n,n);
for i=n-1:-1:1
 x(i)=(b(i)-U(i,i+1)*x(i+1))/U(i,i);
endfor
endfunction
```

```
function x=lap_solve2(n,b)
[L,U]=lu(laplacian(n));
y=inf_solve2(L,b);
x=sup_solve2(U,y);
endfunction
```

Lezione 5

```
function [L,U]=my_lu(A)
 n=size(A)(1);
 L=eye(n);
 U=A;
 for k=1:n
   pivot=U(k,k);
   L(k+1:n,k)=U(k+1:n,k)/pivot;
   colonnaL=L(k+1:n,k);
   rigaU=U(k,k+1:n);
```

```
    U(k+1:n,k+1:n)=U(k+1:n,k+1:n)-colonnaL*rigaU;
  endfor
endfunction
```

```
function x=sys_solve(A,b)
[L,U]=my_lu(A);
y=inf_solve(L,b);
x=sup_solve(U,y);
endfunction
```

```
 function x=sys_solve2(A,b)
  n=size(A)(1);
  U=A;
  y=b;
  for k=1:n
    pivot=U(k,k);
    colonnaL=U(k+1:n,k)/pivot;
    rigaU=U(k,k+1:n);
    U(k+1:n,k+1:n)=U(k+1:n,k+1:n)-colonnaL*rigaU;
    y(k+1:n)=y(k+1:n)-colonnaL*y(k);
    U(k+1:n,k)=0;
  endfor
  %ora U e' triangolare superiore, e y=L^{-1}b
  x=sup_solve(U,y);
endfunction
```

```
function v=householder_vector(x)
%trattiamo solo il caso reale per chiarezza --- modifiche minori per quello complesso
n=length(x);
numeratore=0;
for k=2:n
numeratore=numeratore-x(k)^2; %questa riga va modificata se x puo' essere complesso...
endfor
norma=sqrt(x(1)^2-numeratore); %anche questa e la seguente
denominatore=x(1)+sign(x(1))*norma; %"help sign" per saperne di piu'
v=x;
v(1)=numeratore/denominatore;
endfunction
```

```
function x=qr_solve(A,b);
n=size(A)(1);
R=A;
y=b;
for k=1:n-1
  v=householder_vector(R(k:end,k));
```

```
  %le prossime due righe calcolano H*U(k:end,k:end) e H*y(k:end)
  %non facciamo il prodotto "direttamente" perche'
  %costerebbe O(k^3) per ogni iterazione, quindi O(n^4) complessivamente
  R(k:end,k:end)=R(k:end,k:end)-2*v*(v'*R(k:end,k:end))/norm(v)^2;
  y(k:end)=y(k:end)-2*v*(v'*y(k:end))/norm(v)^2;
endfor
%ora R e' triangolare superiore, e y=Q^{-1}b
x=sup_solve(U,y);
endfunction
```

```
function x=jacobi(A,b,k)
n=size(A)(1);
x=zeros(n,1);
for iteration=1:k
 x_new=zeros(n,1);
 for i=1:n
  x_new(i)=(b(i)-A(i,1:i-1)*x(1:i-1)-A(i,i+1:n)*x(i+1:n))/A(i,i);
 endfor
 x=x_new;
 norm(A*x-b)
endfor
endfunction
```

Lezione 6

```
function x=gaussseidel(A,b,k)
n=size(A)(1);
x=zeros(n,1);
for iteration=1:k
 for i=1:n
  x(i)=(b(i)-A(i,1:i-1)*x(1:i-1)-A(i,i+1:n)*x(i+1:n))/A(i,i);
 endfor
 norm(A*x-b)
endfor
endfunction
```

```
function x=jacobi_opt(A,b)
epsilon=1e-8;
n=size(A)(1);
x=zeros(n,1);
while(norm(A*x-b)>1e-8)
 x_new=zeros(n,1);
 for i=1:n
  x_new(i)=(b(i)-A(i,1:i-1)*x(1:i-1)-A(i,i+1:n)*x(i+1:n))/A(i,i);
 endfor
```

```
 x=x_new;
 norm(A*x-b)
endwhile
endfunction
```

```
function x=jacobi_lap(A,b,k)
n=size(A)(1);
x=zeros(n,1);
for iteration=1:k
 x_new=zeros(n,1);
 %sommiamo solo i termini con A(i,j) non nullo
 %eseguiamo i casi i=1 e i=n a mano, perche' le formule sono diverse
 %(c'e' un termine in meno da sommare...)
 x_new(1)=(b(1)-A(1,2)*x(2))/A(1,1);
 for i=2:n-1
  x_new(i)=(b(i)-A(i,i-1)*x(i-1)-A(i,i+1)*x(i+1))/A(i,i);
 endfor
 x_new(n)=(b(n)-A(n,n-1)*x(n-1))/A(n,n);
 x=x_new;
 norm(A*x-b)
endfor
endfunction
```

Lezione 7

```
function z=membrana(bordo,carico,r)
%usa Gauss-Seidel
m=size(bordo)(1);
n=size(bordo)(2);
z=bordo;
%la seguente e le istruzioni che usano "residuo"
%servono a testare se tutto funziona:
%misurano norm(A*x-b) per il sistema lineare che stiamo risolvendo
%e lo stampano a schermo dopo ogni iterazione

residuo=zeros(m,n);
for iteration=1:r
 %passo di Gauss-Seidel
 for i=2:m-1
  for j=2:n-1
   z(i,j)=-1/4*(carico(i,j)-z(i-1,j)-z(i+1,j)-z(i,j-1)-z(i,j+1));
  endfor
 endfor

 %testiamo il residuo
```

```
 for i=2:m-1
  for j=2:n-1
    residuo(i,j)=carico(i,j)-z(i-1,j)-z(i+1,j)-z(i,j-1)-z(i,j+1)+4*z(i,j);
  endfor
 endfor
 norm(residuo)
endfor
```

```
function Z=membrana_multigrid(Bordo,Carico,k,r)
%le lettere maiuscole si riferiscono alla griglia piu' fitta
%quelle minuscole a quella piu' rada
M=2^(k+1)+1;
N=2^(k+1)+1;
m=2^k+1;
n=2^k+1;
%restrizione
%prende un punto si' e uno no
%ci sono funzioni di restrizione migliori di questa...
bordo=Bordo(1:2:M,1:2:N);
carico=Carico(1:2:M,1:2:N);
%soluzione del problema piu' piccolo
%se possiamo con multigrid
if(k>1)
z=membrana_multigrid(bordo,carico,k-1,r);
else
z=membrana(bordo,carico,r);
end
%estensione: piu' complicata
%sappiamo gia' i punti con coordinate dispari
%ci serve calcolare quelli con una o entrambe
%le coordinate pari
Z(1:2:M,1:2:N)=z; %punti (d,d) gia' noti
 %(p,d) mediando i due (d,d) che gli stanno vicini
Z(2:2:M-1,1:2:N)=1/2*(Z(1:2:M-2,1:2:N)+Z(3:2:M,1:2:N));
%(d,dp) idem
Z(1:2:M,2:2:N-1)=1/2*(Z(1:2:M,1:2:N-2)+Z(1:2:M,3:2:N));
%(p,p) mediando i 4 punti che gli stanno vicini
%(in realta' si possono riciclare un po' di conti dai casi prima...)
Z(2:2:M-1,2:2:N-1)=1/4*(Z(1:2:M-2,1:2:N-2)+Z(1:2:M-2,3:2:N)+Z(3:2:M,1:2:N-2)+Z(3:2:M,3:2:N));

%ora Z (valore iniziale) e' a posto, ma il bordo va risistemato.
Z(1,:)=Bordo(1,:);
Z(M,:)=Bordo(M,:);
Z(:,1)=Bordo(:,1);
Z(:,N)=Bordo(:,N);
```

```
%controlliamo "quanto bene va" questa approssimazione
%misurando norm(A*x-b)
residuo=zeros(M,N);
for i=2:M-1
 for j=2:N-1
  residuo(i,j)=Carico(i,j)-Z(i-1,j)-Z(i+1,j)-Z(i,j-1)-Z(i,j+1)+4*Z(i,j);
 endfor
endfor
norm(residuo);

for iteration=1:r %iterazioni di GS sulla griglia grossa
 for i=2:M-1
  for j=2:N-1
   Z(i,j)=-1/4*(Carico(i,j)-Z(i-1,j)-Z(i+1,j)-Z(i,j-1)-Z(i,j+1));
  endfor
 endfor
 %calcola il residuo
 for i=2:M-1
  for j=2:N-1
   residuo(i,j)=Carico(i,j)-Z(i-1,j)-Z(i+1,j)-Z(i,j-1)-Z(i,j+1)+4*Z(i,j);
  endfor
 endfor
 norm(residuo)
endfor
endfunction
```

Lezione 8

```
function y=horner(p,x)
n=length(p);
y=p(1);
for i=2:n
 y=y*x+p(i);
endfor
endfunction
```

```
function dp=derivata(p)
n=length(p);
dp=zeros(1,n-1);
for i=1:n-1
dp(i)=(n-i)*p(i);
endfor
endfunction
```

```
function x=newton(p,x0);
  x=x0;
  px=horner(p,x0);
  dp=polyderiv(p);
  while(abs(px)>1E-12)
    x=x-px/polyval(dp,x);
    px=polyval(p,x);
  endwhile
endfunction
```

```
function val=decidi(x)
 d1=norm(x-1);
 d2=norm(x-(1+sqrt(3)*I)/2);
 d3=norm(x-(1-sqrt(3)*I)/2);
 [inutile, val]=min([d1 d2 d3]);
endfunction
```

```
function img=newtonfractal()
range=-2:0.04:2;
poly=[1 0 0 -1];
img=zeros(101,101);
for i=1:101
 for j=1:101
  z0=range(i)+1I*range(j);
  rad=newton(poly,z0);
  img(i,j)=decidi(rad);
 endfor
endfor
endfunction
```